FORM PTO-1390 (REV 10-2000)	U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE	ATTORNEY'S DOCKET NUMBER 1968NP/C503326-003
TRANSMITTAL LETTER TO THE UNITED STATES DESIGNATED/ELECTED OFFICE (DO/EO/US) CONCERNING A FLILING UNDER 35 U.S.C. 371		U.S. APPLICATION NO. (If known, see 37 CFR 1 5) **09/719399**

INTERNATIONAL APPLICATION NO. PCT/NZ99/00081	INTERNATIONAL FILING DATE 10 June 1999 (10.06.99)	PRIORITY DATE CLAIMED 10 June 1998 (10.06.98)

TITLE OF INVENTION

SOFTWARE WATERMARKING TECHNIQUES

APPLICANT(S) FOR DO/EO/US

Christian Sven Collberg and Clark David Thomborson

Applicant herewith submits to the United States Designated/Elected Office (DO/EO/US) the following items and other information:

1. ☑ This is a **FIRST** submission of items concerning a filing under 35 U.S.C. 371.
2. ☐ This is a **SECOND** or **SUBSEQUENT** submission of items concerning a filing under 35 U.S.C. 371.
3. ☐ This is an express request to promptly begin national examination procedures (35 U.S.C. 371(f)).
4. ☑ The US has been elected by the expiration of 19 months from the priority date (PCT Article 31).
5. ☑ A copy of the International Application as filed (35 U.S.C. 371(c)(2))
   a. ☐ is attached hereto (required only if not communicated by the International Bureau
   b. ☑ has been communicated by the International Bureau.
   c. ☐ is not required, as the application was filed in the United States Receiving Office (RO/US).
6. ☐ An English language translation of the International Application as filed (35 U.S.C. 371(c)(2)).
7. ☐ Amendments to the claims of the International Application under PCT Article 19 (35 U.S.C. 371(c)(3))
   a. ☐ are attached hereto (required only if not communicated by the International Bureau).
   b. ☐ have been communicated by the International Bureau.
   c. ☐ have not been made; however, the time limit for making such amendments has NOT expired.
   d. ☐ have not been made and will not be made.
8. ☐ An English language translation of the amendments to the claims under PCT Article 19 (35 U.S.C. (c)(3)).
9. ☑ An Oath or Declaration of the inventor(s) (35 U.S.C. (c)(4)). **Copy - unexecuted**
10. ☐ An English language translation of the annexes to the International Preliminary Examination Report under PCT Article 36 (35 U.S.C. 371(c)(5)).

Items 11 to 16 below concern document(s) or information included:

11. ☐ An Information Disclosure Statement under 37 CFR 1.97 and 1.98.

12. ☐ An assignment document for recording. A separate cover sheet in compliance with 37 CFR 3.28 and 3.31 is included.

13. ☑ A **FIRST** Preliminary Amendment.
   ☐ A SECOND or SUBSEQUENT Preliminary Amendment.

14. ☐ A substitute specification.

15. ☐ A change of Power of Attorney and/or address letter.

16. ☑ Other items or information: **Copy of Preliminary Examination Report with Annexes of replacement pages 5, 18 and 20.**

JC05 Rec'd PCT/PTO 0 8 DEC 2000

U.S. APPLICATION NO. (if known, See 37 CFR 1.5) 09/719399	INTERNATIONAL APPLICATION NO. PCT/NZ99/00081	ATTORNEY' DOCKET NUMBER 1968NP/C503326-003

17. ☑ The following fees are submitted	CALCULATIONS	PTO USE ONLY

BASIC NATIONA FEE (37 CFR 1.482)

Neither international preliminary examination fee (37 CFR 1.482)
nor international search fee (37 CFR 1.445(a)(2)) paid to USPTO $1,000.00
and International Search Report not prepared by the EPO or JPO

International preliminary examination fee (37 CFR 1.482) not paid to
USPTO but International Search Report prepared by the EPO or JPO . . . . . .$ 860.00

International preliminary examination fee (37 CFR 1.482) not paid to
USPTO but international search fee (37 CFR 1.445(a)(2)) paid to USPTO. . $ 710.00

International preliminary examination fee paid to USPTO (37 CFR 1.482)
but all claims did not satisfy provisions of PCT Article 33 (1)-(4). . . . .$ 690.00

International preliminary examination fee paid to USPTO (37 CFR 1.482)
and all claims satisfied provisions of PCT Article 33 (1)-(4) . . . .$ 100.00

	CALCULATIONS	PTO USE ONLY
ENTER APPROPRIATE BASIC FEE AMOUNT	$ 1,000.00	
Surcharge of **$130.00** for furnishing the Oath or Declaration later than ☐ 20 ☑ 30 months from the earliest claimed priority date (37 CFR 1.492(e)).	$ 130.00	

CLAIMS	NUMBER FILED	NUMBER EXTRA	RATE		
Total Claims	57 - 20	37	x 18.00	$ 666.00	
Independent Claims	16 - 3	13	x 80.00	$ 1,040.00	
MULTIPLE DEPENDENT CLAIM(S) (if applicable)			+ 270.00	$ 270.00	
TOTAL OF ABOVE CALCULATIONS =				$ 3,106.00	
☐ Applicant claims Small Entity Status. See 37 CFR 1.27. Fees reduced by 50%.				$	
SUBTOTAL =				$ 3,106.00	
Surcharge of **$130.00** for furnishing the English translation later than ☐ 20 ☐ 30 months from the earliest claimed priority date (37 CFR 1.492(f)).				$	
TOTAL NATIONAL FEE =				$ 3,106.00	
Fee for recording the enclosed assignment (37 CFR 1.21(h)). The assignment must be accompanied by an appropriate cover sheet (37 CFR 3.28, 3.31). **$40.00** per property. +				$	
TOTAL FEES ENCLOSED =				$ 3,106.00	
			Amount to be refunded:	$	
			charged:	$	

a. ☑ Check No. 1809 in the amount of $ 3,106.00 for payment of the above fees is enclosed.

b. ☐ Please charge Deposit Account No. _____ in the amount of $_____ for payment of the above fees.
A duplicate of this sheet is attached.

c. ☑ The Commissioner is hereby authorized to charge any fees or credit any overpayment associated with this communication to Deposit Account 02-2120 (Sawyer Law Group LLP).

NOTE: Where an appropriate time limit under 37 CFR 1.494 or 1.495 has not been met, a petition to revive (37 CFR 1.137(a) or (b) must be filed and granted to restore the application to pending status.

SEND ALL CORRESPONDENCE TO: .

Joseph A. Sawyer, Jr.
Sawyer Law Group LLP
P.O. Box 51418
Palo Alto, California 94303
US

Joseph A. Sawyer, Jr., Reg. No. 30,801

12/3/00

Date

JC05 Rec'd PCT/PTO   0 8 DEC 2000

U.S. APPLICATION NO. (If known, see 37 CFR 1.5) 09/719399	INTERNATIONAL APPLICATION NO. PCT/NZ99/00081	ATTORNEY' DOCKET NUMBER 1968NP/C503326-003

17. ☑ The following fees are submitted

BASIC NATIONA FEE (37 CFR 1.482)

	CALCULATIONS	PTO USE ONLY

Neither international preliminary examination fee (37 CFR 1.482)
nor international search fee (37 CFR 1.445(a)(2)) paid to USPTO $1,000.00
and International Search Report not prepared by the EPO or JPO

International preliminary examination fee (37 CFR 1.482) not paid to
USPTO but International Search Report prepared by the EPO or JPO . . . . . .$ 860.00

International preliminary examination fee (37 CFR 1.482) not paid to
USPTO but international search fee (37 CFR 1.445(a)(2)) paid to USPTO. . $ 710.00

International preliminary examination fee paid to USPTO (37 CFR 1.482)
but all claims did not satisfy provisions of PCT Article 33 (1)-(4). . . . .$ 690.00

International preliminary examination fee paid to USPTO (37 CFR 1.482)
and all claims satisfied provisions of PCT Article 33 (1)-(4) . . . .$ 100.00

ENTER APPROPRIATE BASIC FEE AMOUNT	$ 1,000.00	
Surcharge of **$130.00** for furnishing the Oath or Declaration later than ☐ 20 ☑ 30 months from the earliest claimed priority date (37 CFR 1.492(e)).	$ 130.00	

CLAIMS	NUMBER FILED	NUMBER EXTRA	RATE	$	
Total Claims	57 - 20	37	x 18.00	$ 666.00	
Independent Claims	16 - 3	13	x 80.00	$ 1,040.00	
MULTIPLE DEPENDENT CLAIM(S) (if applicable)			+ 270.00	$ 270.00	

TOTAL OF ABOVE CALCULATIONS =	$ 3,106.00	
☐ Applicant claims Small Entity Status. See 37 CFR 1.27. Fees reduced by 50%.	$	
SUBTOTAL =	$ 3,106.00	
Surcharge of **$130.00** for furnishing the English translation later than ☐ 20 ☐ 30 months from the earliest claimed priority date (37 CFR 1.492(f)).	$	
TOTAL NATIONAL FEE =	$ 3,106.00	
Fee for recording the enclosed assignment (37 CFR 1.21(h)). The assignment must be accompanied by an appropriate cover sheet (37 CFR 3.28, 3.31). **$40.00** per property. +	$	
TOTAL FEES ENCLOSED =	$ 3,106.00	
	Amount to be refunded:	$
	charged:	$

a. ☑ Check No. 1809 in the amount of **$ 3,106.00** for payment of the above fees is enclosed.

b. ☐ Please charge Deposit Account No. _____ in the amount of $_____ for payment of the above fees. A duplicate of this sheet is attached.

c. ☑ The Commissioner is hereby authorized to charge any fees or credit any overpayment associated with this communication to Deposit Account 02-2120 (Sawyer Law Group LLP).

NOTE: Where an appropriate time limit under 37 CFR 1.494 or 1.495 has not been met, a petition to revive (37 CFR 1.137(a) or (b) must be filed and granted to restore the application to pending status.

SEND ALL CORRESPONDENCE TO:

Joseph A. Sawyer, Jr.
Sawyer Law Group LLP
P.O. Box 51418
Palo Alto, California 94303
US

Joseph A. Sawyer, Jr., Reg. No. 30,801

12/5/00
Date

Attorney Docket: 1968NP/CC503326-003

JC05 Rec'd PCT/PTO   0 8 DEC 2000

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re Application of:	Date: December 8, 2000
Collberg, et al.	
Serial No.: Unassigned	Group Art Unit: Unassigned
Filed: Herewith	Examiner: Unassigned

For: SOFTWARE WATERMARKING TECHNIQUES

Honorable Commissioner of Patents and Trademarks
Washington, D.C. 20231

PRELIMINARY AMENDMENT

Sir:

Please amend the above-identified application in the following manner:

IN THE CLAIMS:

1. (Amended) A method of watermarking a software object comprising the steps of:
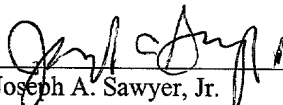
   (a)   providing an input sequence; and

   (b)   storing [whereby] a watermark [is stored] in the state of the software object as [it] the software object is being run with [a particular] the input sequence.

2. (Amended) [A] The method as claimed in claim 1 wherein the software object [may be] is a program or a piece of a program.

3. (Amended) [A] The method as claimed in [any one of] claim[s] 1 [or 2] wherein the state of the software object [may] corresponds to the current values held in [the]a stack, a heap, and global variables [, registers, program counter and the like,]of the software object.

4. (Amended) [A] The method [as claimed in any preceding] of claim 1 or 2 or 3 wherein the watermark is stored in an [object's] execution state of the software object whereby [an]the input sequence [I] is constructed which, when fed to an application of which the software object is a part, will make the software object [O] enter a second state which [represents]is a representation of the watermark, the representation being validated or checked by examining the stack, heap, global variables, registers, program counter and the like, of the software object [O].

5. (Amended) [A] The method [as claimed in any one] of claim[s] 1 or 2 wherein the watermark is embedded in [the] an execution trace of the software object [O] whereby, as a special input [I] is fed to [O]the software object, [the] an address/operator trace is monitored and, based on a property of the trace, [a]the watermark is extracted.

6. (Amended) [A] The method [as claimed in any one] of claim[s] 1 [to 4] or 2 or 3 wherein the watermark is embedded in [the] a topology of a dynamically built graph structure.

7. (Amended) [A] The method [as claimed in] of claim 6 wherein the dynamically built graph structure [(or watermark graph)] corresponds to a representation of [the]a data structure of the program and may be viewed as a set of nodes together with a set of vertices.

8. (Amended) [A] The method [as claimed in any preceding] of claim 1, or 2 or 3 further comprising the step of:

_____(c)_____building a recognizer [R] concurrently with the input [I] sequence and the watermark [W].

9. (Amended) [A] The method [as claimed in] of claim 8 wherein [R] the recognizer is a function adapted to identify and extract the watermark [graph] from all other dynamic structures on [the] a heap or stack.

10. (Amended) [A] The method [as claimed in either] of claim 8 [or 9] wherein the watermark [W] incorporates a marker that will allow [R] the recognizer to recognize it easily.

11. (Amended) [A] The method [as claimed in any one] of claim[s] 8 [to 10] wherein [R] the recognizer is retained separately from the program and whereby [R] the recognizer inspects the state of the program.

12. (Amended) [A] The method [as claimed in any one] of claim[s] 8 [to 11] wherein [R] the recognizer is dynamically linked with the program when it is checked for the existence of a watermark.

13. (Amended) [A] The method [as claimed in any preceding] of claim 1, or 2, or 3 wherein [the application of which] the software object [forms a] is part of an application that is obfuscated or incorporates tamper-proofing code.

14. (Amended) [A] The method [as claimed in any one] of claim[s] 8 [to 12] wherein [R] the recognizer checks [W] the watermark for a signature property [s(W)].

15. (Amended) [A] The method [as claimed in] of claim 14 wherein the signature property [s(W)] is evaluated by testing for a specific result from a hard computational problem.

16. (Amended) [A] The method [as claimed in either] of claim 14 [or claim 15] including the step of:

_____(d)_____ [the creation of] creating a number having at least one numeric property[n] which [may be] is embedded in the topology of [W] the watermark, whereby the signature property [may be] is evaluated by testing [one or more] the at least one numeric property[properties of n].

17. (Amended) [A] The method [as claimed in] of claim 16 wherein the signature property is evaluated by testing whether [n] the number is [the] a product of two primes.

18. (Amended) A method of verifying the integrity or origin of a program including the steps of:

(a)_____watermarking the program with a watermark [W], wherein the watermark [W] is stored in the state of a program as the program is being run with an [particular] input sequence[I];

(b)_____building a recognizer [R] concurrently with the input [I] sequence and watermark W wherein the recognizer is adapted to extract the watermark [graph] from other dynamically allocated data wherein [R] the recognizer is kept separately from the program; wherein [R]the

recognizer is adopted to check for a number[*n*].

19. (Amended) [A] The method [of verifying the integrity of origin of a program as claimed in] of claim 18, wherein the number [*n*] is the product of two primes and wherein [*n*] the number is embedded in [the] a topology of [*W*] the watermark.

20. (Amended) [A] The method [as claimed in either] of claim 18 or claim 19 wherein the number [*n*] is derived from a[ny] combination of numbers depending on [the] a context and an application for the watermark.

21. (Amended) [A] The method [as claimed in any one] of claim[s] 18 or 19 [to 20] wherein the program [or code] is further adapted to be resistant to tampering, the resistance to tampering capable of being [preferably]by means of obfuscation or by adding tamper-proofing code.

22. (Amended) [A] The method [as claimed in any one] of claim[s] 18 or 19 [to 21] wherein the recognizer [*R*] checks for the effect of the watermark[ing code] on [the] an execution state of the [application]program, thereby preserving [the] an ability to recognize the watermark [in cases] where semantics-preserving transformations have been applied to the [application]program.

23. (Amended) A method of watermarking software including the steps of:

(a)   embedding a watermark in a static string; and

(b)   applying an obfuscation technique whereby [this]the static string is converted into executable code.

24. (Amended) A method of watermarking software comprising the steps of:

(a)   [wherein the]choosing a watermark [W is chosen] from a class of graphs [G]having a plurality of members, [wherein] each member of [G] the class of graphs has [one or more properties, such as planarity] at least one property, [said] the at least one property being capable of being tested by integrity-testing software; and

(b)   applying the watermark to the software.

25. (Amended) [A] The method of [watermarking software as claimed in] claim 24 wherein the watermark is rendered tamperproof to certain transformations by subjecting the watermark graph to one or more local transformations.

26. (Amended) [A] The method of [watermarking software as claimed in] claim 25 wherein the watermark is a watermark graph including at least one node and wherein each of the at least one node of the watermark graph is expanded into a cycle.

27. (Amended) A method of fingerprinting software comprising the steps of:

      (a)   providing [wherein] a plurality of watermarked programs, [obtained as claimed in any preceding claim are produced] the plurality of watermarked programs being obtained by providing an input sequence for each program of the plurality of programs and storing a watermark in a state of a software object for the program as the software object is being run with

the input sequence.

28. (Amended) [A]The method of fingerprinting software as claimed in claim 27 wherein the plurality of watermarked programs each of which has a number [n] with a common prime factor [p].

Please cancel claim 29.

30. (Amended) [Software written to perform the method as claimed in any preceding claim]A computer readable medium including a program for watermarking a software object, the program including instructions for:

(a)   providing an input sequence; and

(b)   storing a watermark in the state of the software object as the software object is being run with the input sequence.

31. (Amended) A computer [programmed to perform the method as claimed in any one of claims 1 to 27] comprising:

a software object;

an input sequence; and

a watermark stored in the state of the software object as the software object is being run with the input sequence.

Please add the following claims:

32.   A method of fingerprinting software comprising the steps of:

(a)   providing a plurality of watermarked programs, the plurality of watermarked

programs being obtained by watermarking each program of the plurality of programs with a

watermark, wherein the watermark is stored in the state of a program as the program is being run

with an input sequence and building a recognizer concurrently with the input sequence and

watermark W wherein the recognizer is adapted to extract the watermark from other dynamically

allocated data wherein the recognizer is kept separately from the program; wherein the

recognizer is adapted to check for a number.

33.   A method of fingerprinting software comprising the steps of:

(a)   providing a plurality of watermarked programs, the plurality of watermarked programs

being obtained by watermarking each program of the plurality of programs with a watermark, the

watermark being obtained by embedding a watermark in a static string and applying an obfuscation

technique whereby the static string is converted into executable code.

34.   A method of fingerprinting software comprising the steps of:

(a)   providing a plurality of watermarked programs, the plurality of watermarked programs

being obtained by watermarking each program of the plurality of programs with a watermark, the

watermark being obtained by choosing a watermark from a class of graphs having a plurality of

members, each member of the class of graphs has at least one property, the at least one property

being capable of being tested by integrity-testing software and applying the watermark to the

software.

35. A computer-readable medium including a program for verifying the integrity or origin of a program, the program including instructions for:

(a) watermarking the program with a watermark, wherein the watermark is stored in the state of a program as the program is being run with an input sequence;

(b) building a recognizer concurrently with the input sequence and watermark wherein the recognizer is adapted to extract the watermark from other dynamically allocated data wherein the recognizer is kept separately from the program; wherein the recognizer is adapted to check for a number.

36. A computer-readable medium including a program for watermarking software, the program including instructions for:

(a) embedding a watermark in a static string; and

(b) applying an obfuscation technique whereby the static string is converted into executable code.

37. A computer-readable medium including a program for watermarking software, the program including instructions for:

(a) choosing a watermark from a class of graphs having a plurality of members, each member of the class of graphs has at least one property, the at least one property being capable of being tested by integrity-testing software; and

(b) applying the watermark to the software.

38. A computer capable of verifying the integrity or origin of a program, the computer

comprising:

an input sequence;

a watermark for watermarking the program, wherein the watermark is stored in the state of a program as the program is being run with the input sequence;

a recognizer built concurrently with the input sequence and watermark wherein the recognizer is adapted to extract the watermark from other dynamically allocated data wherein the recognizer is kept separately from the program; wherein the recognizer is adapted to check for a number.

39.   A computer for watermarking software comprising:

a static string;

a watermark embedded in the static string; and

an obfuscation technique for converting the static string into executable code.

40.   A computer comprising:

a watermark from a class of graphs having a plurality of members, each member of the class of graphs has at least one property, the at least one property being capable of being tested by integrity-testing software; and

software to which the watermark is applied.

09/719399

IN THE SPECIFICATION:

Please add the following Abstract:

ABSTRACT
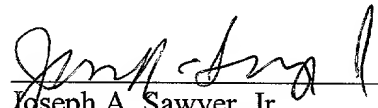
A method and system for watermarking software is disclosed. In one aspect, the method and system include providing an input sequence and storing a watermark in the state of a software object as the software object is being run with the input sequence. In another aspect, the method and system verify the integrity or origin of a program by watermarking the program. The watermark is stored as described above. In this aspect, the method and system also include building a recognizer concurrently with the input sequence and the watermark. The recognizer can extract the watermark from other dynamically allocated data and is kept separately from the program. The recognizer is adapted to check for a number. In another aspect, the software is watermarked by embedding a watermark in a static string and applying an obfuscation technique to convert the static string into executable code. In another aspect, the watermark is chosen from a class of graphs having a plurality of members and applied to the software. Each member of the class of graphs has at least one property that is capable of being tested by integrity-testing software.

-11-

REMARKS

This Amendment is made to more particularly claim the present invention. Claims 1-31 are pending in the international application, Serial No. PCT/NZ99/00081. In the present national phase application, Applicant has amended claims 1-28 and 30-31, and has canceled claim 29. Applicant has also added claims 32-40. Consequently, claims 1-28 and 30-40 remain pending in the present application entered herewith into the national phase under 37 USC 3.71.

Applicant's attorney believes that this application is in condition for allowance. Should any unresolved issues remain, Examiner is invited to call Applicant's attorney at the telephone number indicated below.

Respectfully submitted,

Joseph A. Sawyer, Jr.
Sawyer Law Group LLP
Attorney for Applicant
(650) 493-4540

1

SOFTWARE WATERMARKING TECHNIQUES

FIELD OF THE INVENTION

The present invention relates to methods for protecting software against theft, establishing/proving ownership of software and validating software. More particularly, although not exclusively, the present invention provides for methods for watermarking what will be generically referred to as software objects. In this context, software objects may be understood to include programs and certain types of media.

BACKGROUND TO THE INVENTION

Watermarking is the process of embedding a secret message, the *watermark*, into a cover or overt message. For example, in media watermarking, the secret is commonly a copyright notice and the cover is a digital image, video or audio recording. Fingerprinting is a method whereby each individual software application incorporates a, potentially, unique, watermark which allows that particular example of the software to be identified. Fingerprinting may be viewed as a multiple use of watermarking techniques.

The watermark is constructed to make it difficult to remove the watermark without damaging the software object in which it is embedded. Such watermarks may only be removed safely by someone (or some process) in possession of one or more secrets that were employed while constructing the watermark.

Watermarking a software object (hereafter referred to as an *object*) discourages intellectual property theft. A further application is that watermarking an object can be used to establish and/or prove evidence of ownership of an object. Fingerprinting is similar to watermarking except a different watermark is embedded in every cover message thus providing a unique fingerprint for every object. Watermarking is therefore a subset of fingerprinting and the latter may be used to detect not only the fact that a theft has occurred, but may also allow identification of the particular object and thus establish an audit trail which can be used to reveal the infringer of copyright.

In the context of prior art watermark techniques, the following scenario serves to illustrate the ways in which a watermarked object may be vulnerable to attack. With

reference to figure 1, suppose that A watermarks an object *O* with a watermark *W* and key *K*. If the object *O* is sold to B and B wishes to (illegally) on-sell *O* to C, there are various types of attack to which *O* may be vulnerable.

5 *Detection:* initially B must try and detect the presence of the watermark in *O*. If there is no watermark, no further action is necessary.

Locate and remove: once B has determined that *O* carries a watermark, B may try to locate and remove *W* without otherwise harming the rest of the contents of *O*.

10

Distort: if some degradation in quality of *O* is acceptable, B may distort it sufficiently so that it becomes impossible for A to detect the presence of the watermark *W* in the object *O*.

15 *Add:* alternatively, if removing the watermark *W* is too difficult, or distorting the object *O* is not acceptable, B might simply add his own watermark *W'* (or several such marks) to the object *O*. This way, A's mark becomes just one of many.

It is considered that most media watermarking schemes are vulnerable to attack by
20 distortion. For example, image transforms such as cropping and lossy compression will distort the image sufficiently to render many watermarks unrecoverable.

To the knowledge of the applicants there exists no effective watermarking scheme which is capable of use with or appropriate for software. It would be a significant
25 advantage to be able to apply watermarking techniques to software in view of the widespread occurrence of software piracy. It is estimated at software piracy costs approximately 15 billion dollars per year. Thus the problem of software security and protection is of significant commercial importance.

30 One simple way, known in the prior art, of embedding a watermark in a piece of software is simply to include it in the initialized static data section of the object code. In a similar, yet more complex manner, watermarks are often encoded in what is known as an "Easter egg". This is a piece of code, which is activated for a highly unusual or seldom encountered input to the particular application, which displays a

watermark image, plays a watermark sound, or, in some way, alerts the user that the watermark code has been activated.

Thus, it is an object of the present invention to provide methods for watermarking software objects which overcomes the limitations inherent in prior art watermarking techniques and allows for non-media objects to be watermarked effectively. It is a further object of the present invention to provide methods for watermarking software objects which are resistant to the aforementioned techniques for attacking watermark objects or to at least provide the public with a useful choice.

DISCLOSURE OF THE INVENTION

In one aspect, the invention provides for a method of watermarking a software object whereby a watermark is stored in the state of the software object as it is being run with a particular input sequence.

The software object may be a program or piece of program.

The state of the software object may correspond to the current values held in the stack, heap, global variables, registers, program counter and the like.

In a preferred embodiment, the watermark may be stored in an object's execution state whereby a (possibly empty) input sequence I is constructed which, when fed to an application of which the object is a part, will make the object O enter a state which represents the watermark, the representation being validated or checked by examining the dynamically allocated data structures of the object O.

In an alternative embodiment, the watermark could be embedded in the execution trace of the object O whereby, as a special input I is fed to O, the address/operator trace is monitored and, based on a property of the trace, a watermark is extracted.

In a preferred embodiment, the watermark is embedded in the state of the program as it is being run with a particular input sequence $I=I_1... I_k.$

The watermark may be embedded in the topology of a dynamically built graph structure.

4

The graph structure (or watermark graph) corresponds to a representation of the data structure of the program and may be viewed as a set of nodes together with a set of vertices.

The method may further comprise building a recognizer R concurrently with the input I and watermark W.

Preferably R is a function adapted to identify and extract the watermark graph from all other dynamically allocated data structures.

In an alternative, less preferred embodiment, the watermark W may incorporate a marker that will allow R to recognize it easily.

In a preferred embodiment, R is retained separately from the program whereby R is dynamically linked with the program when it is checked for the existence of a watermark.

Preferably the application of which the object forms a part is obfuscated or incorporates tamper-proofing code.

In a preferred embodiment, R extracts a value n from the topology of the graph comprising the watermark W.

The watermark W has a signature property s where $s(W)$ evaluates to "true" if the watermark W is recognisable wherein the recogniser R tests a presumed watermark W' by evaluating the signature property $s(W')$.

In a preferred embodiment, the method includes the creation of a number n which may be embedded in the topology of a watermark graph, wherein the signature property $s(W)$ is a function of a number n so embedded.

In a preferred embodiment, the signature property $s(W)$ is "true" if and only if the number n is the product of two primes.

The invention further provides for a method of verifying the integrity or origin of a program including:

watermarking the program with a watermark W in the state of a program as the program is being run with a particular input sequence I;

building a recognizer R concurrently with the input I and watermark W wherein the recognizer is adapted to extract the watermark graph from other dynamically allocated data structures wherein R is kept separately from the program; wherein R is adapted to check for a number n, n, in a preferred embodiment, being the product of two primes and wherein n is embedded in the topology of W.

Preferably, the signature property may be evaluated by testing for a specific result from a hard computational problem.

The number n may be derived from any combination of numbers depending on the context and application.

Preferably the program or code is further adapted to be resistant to tampering, preferably by means of obfuscation or by adding tamper-proofing code.

Preferably the watermarks W are chosen from a class of graphs G wherein each member of G has one or more properties, such as planarity, said property being capable of being tested by integrity-testing software.

In an alternative embodiment, the watermark may be rendered tamperproof to certain transformations, such as attacks, by expanding each node of the watermark graph into a j-cycle, where j may be any number from 1 to 5.

In a broad aspect, the recognizer R checks for the effect of the watermarking code on the execution state of the application thereby preserving the ability to recognize the watermark in cases where semantics-preserving transformations have been applied to the application.

In a further aspect, the invention provides for a method of watermarking software including the steps of:

embedding a watermark in a static string, then applying an obfuscation technique whereby this static string is converted into executable code.

The executable code is called whenever the static string is required by the program.

5

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will now be described by way of example only and with

10   reference to the figures in which:

Figure 1:   illustrates methods of adding a watermark to an object and attacking
                the integrity of such a watermark;

15   Figure 2:   illustrates methods of embedding a watermark in a program;

Figure 3:   illustrates an example of a function used to embed a watermark within
                a static string;

20   Figure 4:   illustrates insertion of a bogus predicate into a program;

Figure 5:   illustrates splitting variables;

Figure 6:   illustrates merging variables;

25

Figure 7:   illustrates the conversion of a code section into a different virtual
                machine code;

Figure 8:   illustrates an example of a method of the watermarking scheme
30   according to the present invention;

Figure 9:   illustrates a possible encoding method for embedding a number in the
                topology of a graph;

Figure 10:   illustrates another possible embodiment for embedding a number in the topology of a graph;

Figure 11;   illustrates a marker in a graph;

Figure 12:   illustrates examples of obfuscating transformations;

Figure 13:   illustrates examples of tamperproofing Java code;

Figure 14:   illustrates enumeration encoding in a planted plane cubic tree on $2m =$ 8 nodes; and

Figure 15:   illustrates tamperproofing against node-splitting.

Referring to Figure 1(b) a way is shown by which Bob can circumvent a watermarking scheme by distorting the protected object. If the distortion is at ``just the right level", O will still be usable by Bob, but Charles will be unable to extract the watermark. In Figure 1(9), the distortion is so severe that O is no longer functional, so Bob will not be able to use it, nor is he able to on-sell it.

In the present context, tamperproofing is applied in order to prevent an adversary from removing the watermark and to provide assurance to the software end-user that the software object hasn't been tampered with. Thus the 'integrity' of the program may be verified. The primary aim of the present invention is to allow accurate assertion of ownership of a software object with a secondary purpose being to ensure the integrity of the object.

It has been shown that there are transformations, called obfuscating transformations, that will destroy almost any kind of program structure while preserving the semantics (operational behaviour) of the program. Other semantics preserving transformations, such as optimising transformations known from the prior art can be used to similar

5   effect. As a consequence, any software watermarking technique must be evaluated with respect to its resilience to attack from automatic application of semantics preserving transformations, such as obfuscation. The following discussion will survey obfuscating transformations that can be used to destroy software watermarks.

10   In Figure 2a a watermark is embedded within a static string. There are several ways of rendering watermarks unrecogisable, the most effective perhaps by converting static strings into a program that produces the data. As an example, consider the function G in Figure 3. This function was constructed to obfuscate the strings "AAA", "BAAAA", and "CCB". The values produced by G are G(1)="AAA", G(2)="BAAAA",

15   G(3)=G(5)="CCB", and G(4)="XCB".

In Figure 2b Alice embeds a watermark within the program code itself. There are numerous ways to attack such code. Figure 4, for example, shows how it is possible to insert bogus predicates into a program. These predicates are called opaque since

20   their outcome is known at obfuscation time, but difficult to deduce otherwise. Highly resilient opaque predicates can be constructed using hard static analysis problems such as aliasing.

In Figure 2c a watermark is embedded within the state (global, heap, and stack data,

25   etc.) of the program as it is being run with a particular input I. Different obfuscation techniques can be employed to destroy this state, depending on the type of the data. For example, one variable can be split into several variables (Figure 5) or several variables can be merged into one (Figure 6).

30   In Figure 2d a watermark is embedded within the trace (either instructions or addresses, or both) of the program as it is being run with a special input sequence I = I_1, I_2, ... I_k. In an alternative embodiment, a watermark may be embedded within a series of execution traces, said series of traces being generated as the program is run on a special input. This special input is comprised of a series of one or more

35   input sequences, where each input sequence is generated by a specific process

which may incorporate a random or pseudorandom number generator. Execution traces have many properties that may be observed by a watermark recogniser R. One example of such a property is "if the program passes point P1 in O, then there's a 32% chance that it will also pass point P2". Another example of such a property is

5   the frequency at which some specific basic operation, such as addition, is performed. A specific collection of (one or more) such execution-trace properties is the watermark W. The signature property $s(W)$ for this W is that all the property values are within some predefined tolerance. For example, we might require that our sample property P1-P2 have a value between 30% and 34% on a randomly-

10   generated series of 10000 inputs (note that we would not expect to observe an "exact match" to our 32% estimated mean-value for this property P1-P2, because each randomly-generated series of inputs would give us a somewhat different measurement for this property value).

15   Many of the same transformations that can be used to obfuscate code will also obfuscate an instruction trace. Figure 7 shows another, more potent, transformation. The idea is to convert a section of code (Java bytecode in our case) into a different virtual machine code. The new code is then executed by a virtual machine interpreter included with the obfuscated application. The execution trace of the new virtual

20   machine running the obfuscated program will be completely different from that of the original program. In Figure 2e, a watermark is embedded in an Easter Egg. Unless the code is obfuscated, Easter Eggs may be found by straightforward techniques such as decompilation and disassembly.

25   In this section, techniques for embedding software watermarks in dynamic data structures are discussed. The inventors view these techniques as the most promising for withstanding de-watermarking attacks by obfuscation.

The basic structure of the proposed watermarking technique is outlined in Figure 8.

30   The method is as follows:

1.   The watermark W is embedded, not in the static structure of the program, its code (Unix text segment), its static data (Unix initialised data segment), or its type information (Unix symbol segment or Java's Constant Pool), but rather in

35   the state of the program as it is being run with a particular input sequence I

(of length k) whose elements are $I = I_1, I_2... I_k$. Of course k may be 0, in which case there is no input and the input sequence is empty.

2. More specifically, the watermark is embedded in the topology of a dynamically built graph structure. It is believed that obfuscating the topology of a graph is fundamentally more difficult than obfuscating other types of data. Moreover, it is anticipated that tamperproofing such a structure should be easier than tamperproofing code or static data. This is particularly true of languages like Java, where a program has no direct access to its own code.

3. A Recogniser R is built along with the input I and watermark W. R is a function that is able to identify and extract the watermark graph from among all other dynamic allocated data structures. Since, in general, sub-graph isomorphism is a difficult problem, it is possible that W will have some special marker that will allow R to recognise W easily. Alternatively, W may be formed immediately after input I_k is processed, i.e. markers may not be necessary. Markers are considered 'unstealthy' for the following reason. If a marker is easily recognisable by a recogniser, an adversary might discover it – perhaps by way of a collusive attack on a collection of fingerprinted objects. The use of markers can be avoided by exploiting the recogniser's knowledge of the secret input sequence in the following way: the watermark will be completed immediately after the kth input (I_k) of this sequence is presented to the program. The recogniser knows the value of "k" and therefore is able to look for the watermark graph effectively, by examining the nodes that were allocated or modified during the processing of I_k. In contrast, the adversary would be unaware of the length of this sequence and would therefore have to "guess' a value of "k" as well as the values $(I_1, I_2... I_k)$ in the input sequence I, before looking for the watermark.

4. An important aspect of the proposed technique is that R is not distributed along with the rest of the program. If it were, a potential adversary could identify and decompile it, and discover the relevant property of W. R is employed only when we check for the watermark. R may be an extension of the program comprised of self-monitoring code, or it may be an adjunct to a debugger or some other external means for examining the dynamic state of

the program. R may be linked in dynamically with the program when we check for the watermark. Other mechanisms are envisaged by which the recogniser R may observe the state of the object O.

5.   It is required that some signature property s(W) of W be highly resilient to tampering. This can be achieved, for example, by obfuscation or by adding tamperproofing code to the application.

6.   In Figure 8 it is assumed that the signature that R checks for is a number n, which has been embedded in the topology of W. n is the product of two large primes P and Q. To prove the legal origin of the program, we link in R, run the resulting program with I as input, and show that we can factor the number that R produces. Alternatively, s(W) can be based on hard computational problems other than factorisation of large integers.

The above issues will now be discussed in more detail. The first problem to be solved is how to embed a number in the topology of a graph. There are a number of ways of doing this, and, in fact, a watermarking tool should have a library of many such techniques to choose from. Figure 9 illustrates one possible encoding. The structure is basically a linked list with an extra pointer field which encodes a base-6 digit. A null-pointer encodes a 0, a self-pointer a 0, a pointer to the next node encodes a 1, etc. A further example is shown in figure 14 whereby the watermark W is chosen from a class of graphs G wherein each member of G has one or more properties (in figure 14 – planarity) that may be tested by integrity-checking software. The integrity checking software may be incorporated into the program during the watermarking process.

In the previous paragraph, it was shown how an integer n could be encoded in the topology of a graph. The encoding is resilient to tampering, as long as the recogniser R is able to correctly identify the nodes containing the two pointer fields in which we have encoded n. We now describe another encoding showing that a recogniser R can evaluate n if it can identify only a single pointer field per node.

Using a single pointer per node, we can construct a watermark W in the form of a parent-pointer tree. The parent-pointer tree W is a representation of a graph G

known as an oriented tree enumerable by the techniques described in Knuth, Vol I 3rd Edition, Section 2.3.4.4.

5   The number a_m of oriented trees with m nodes is asymptotically $a_m = c(1/\alpha)^{n-1}/n^{3/2} + O((1/\alpha)^n /n^{5/2})$ for $c \sim 0.44$ and $1/\alpha \sim 2.956$. Thus we can encode an arbitrary 1000-bit integer n in a graphic watermark W with $1000/log_2 2.956 \sim 640$ nodes.

We construct an index n for any enumerable graph in the usual way, that is, by ordering the operations in the enumeration. For example, we might index the trees
10   with m nodes in "largest subtree first" order, in which case the path of length m-1 would be assigned index 1. Indices 2 through a_{m-1} would be assigned to the other trees in which there is a single subtree connected to the root node. Indices $a_{m-1} +1$ through $a_{m-1} + a_{m-2}$ would be assigned to the trees with exactly two subtrees connected to the root node, such that one of the subtrees has exactly m-2 nodes.
15   The next $a_{m-3}a_2 = a_{m-1}$ indices would be assigned to trees with exactly two subtrees connected to the root node, such that one of the subtrees has exactly m-3 nodes. See Figure 10 for an example.

To aid the recognition of a watermark, the recogniser may use secret knowledge of a
20   "signal" indicating that "the next thing that follows" is the real watermark. In a preferred embodiment, the secret is the input sequence I; the recogniser (but not the attacker) knows that the watermark will be constructed after the input sequence $I = I_1, I_2 ... I_k$ has been processed. In an alternative, but less preferred embodiment, the secret is an easily recognisable "marker" that may be present in the watermark
25   graph. This is similar to the signals used between baseball coaches and their players. See Figure 11 for an example.

One advantageous consequence of the present approach is that semantics-preserving transformations, such as those employed in optimising compilers and
30   those employed by obfuscation techniques which target code and static data will have no effect on the dynamic structures that are being built. There are, however, other techniques which can obfuscate dynamic data, and which we will need to tamperproof against. There are three types of obfuscating transformations which will need to be protected against:

35

1. An adversary can add extra pointers to the nodes of linked structures. This will make it hard for R to recognise the real graph within a lot of extra bogus pointer fields.

5 2. An adversary can rename and reorder the fields in the node, again making it hard to recognise the real watermark.

3. Finally, an adversary can add levels of indirection, for example by splitting nodes into several linked parts.

10

These transformations are illustrated in Figure 12. It is important to note here that obfuscating linked structures has some potentially serious consequences. For example, splitting nodes will increase the dynamic memory requirement of the program (each cell carries a certain amount of overhead for type information etc.),

15 which could mean that a program which ran on, say, a machine with 32M of memory would now not run at all. Furthermore, if we assume that an adversary does not know in which dynamic structure our watermark is hidden, he is going to have to obfuscate every dynamic memory allocation in the entire program.

20 Next will be discussed techniques for tamperproofing a dynamic watermark against the obfuscation attacks outlined above.

The types of tamperproofing techniques that will be available will depend on the nature of the distributed object code. If the code is strongly typed and supports

25 reflection (as is the case with Java bytecode) we can use these reflection capabilities to construct the tamperproofing code. If, on the other hand, the application is shipped as stripped, untyped, native code (as is the case with most programs written in C, for example) this possibility is not open to us. Instead, we can insert code which manipulates the dynamically allocated structures in such a way that obfuscating

30 them would be unsafe.

ANSI C's address manipulation facilities and limited reflection capabilities allow for some trivial tamperproofing checks:

35 include <stdlib.h>

```
include <stddef.h>
struct s int a; int b;;
void main ()
        if (offsetof(struct s, a) >
                offsetof(struct s, b)) die();
        if (sizeof(struct s) != 8) die();
}
```

These tests will cause the program to terminate if the fields of the structure are reordered, or the structure is split or augmented.

Figure 13 (a) shows how Java's reflection package allows us to perform similar tamperproofing checks. Note that this example code is not completely general, since Java does not specify the relative order of class fields.

Figure 13 (b) shows how we can also use opaque predicates and variables to construct code which appears to (but in fact, does not) perform ``unsafe" operations on graph nodes. A de-watermarking tool will not be able to statically determine whether it is safe to apply optimising or obfuscating transformations on the code. In the example in Figure 13 (b), V is an opaque string variable whose value is "car", although this is difficult for a de-watermarker to work out statically. At 1 it appears as if some or all (unknown to the de-watermarker) field is being set to null, although this will never happen. The statement 2 is a redundant operation performing n.car = n.car, although (due to the opaque variable R whose value is always 1) this cannot in general be worked out statically.

For increased obscurity, the code to build the watermark should be scattered over the entire application. The only restriction is that when the end of the input sequence $I=I_1 \dots I_k$ is reached, the watermark W has been constructed. This watermark in a preferred embodiment, may be composed of some or all of the *components* $W_1, \dots W_{k-1}$ that were constructed previously. Additionally, in a preferred embodiment, some components W_i may be composed of some of all components constructed before W_i.

$W_0 = ...;$

if (input = I_1) $W_1 = ...;$

if (input = I_2) $W_2 = ...;$

5 if (input = I_{k-1}) $W_{k-1} = ...;$

if (input = I_k) $W = ...;$

In order to identify the watermark structure, the recogniser must be able to enumerate all dynamically allocated data structures. If this is not directly supported

10 by the runtime environment (as, for example, is the case with Java), we have two choices. We can either rewrite the runtime system to give us the necessary functionality or we can provide our own memory allocator. Notice, though, that this is only necessary when we are attempting to recognise the watermark. Under normal circumstances the application can run on the standard runtime system.

15

A further technique is shown in figure 15. Here is illustrated a technique which applies a local transformation, thereby tamperproofing the watermark against an attack by node-splitting. Each of the nodes of the original watermark graph is expanded into a 4-cycle. If an adversary splits two nodes, the underlying structure

20 ensures that these node will fall on a cycle. At (3) the recogniser shrinks the biconnected components of the underlying graphs with the result that the graph is isomorphic to the original watermark.

It is envisaged that local transformations, other than expansion of nodes into cycles,

25 may be employed to tamperproof the watermark against specific attackes other than node-splitting. For example, redundant edges may be introduced into the watermark in order to render the watermark tamperproof to specific attacks which involve the renaming and reordering of fields in nodes.

30 A number of techniques are known in the prior art for hiding copyright notices in the object code of a program. It is the inventors' belief that such methods are not resilient to attack by obfuscation -- an adversary can apply a series of transformations that will hide or obscure the watermark to the extent that it can no longer be reliably retrieved.

35

16

The present invention indicates that the most reliable place to hide a watermark is within the dynamically allocated data structures of the program, as it is being executed with a particular input sequence.

5   A further application for the watermarking technique described above may be in "fingerprinting" software. In this case, each individual program (i.e. every distributed copy of the code) is watermarked with a different watermark. Although there is a risk of an adversary collusively attacking the watermark, the applicant believes that applying obfuscation may render it very difficult for the attacker to interpret the

10   evidence obtained by a collusive attack.

Where in the foregoing description reference has been made to elements or integers having known equivalents, then such equivalents are included as if they were individually set forth.

15

Although the invention has been described by way of example and with reference to particular embodiments, it is to be understood that modifications and/or improvements may be made without departing from the scope or spirit of the invention.

20

CLAIMS:

1. A method of watermarking a software object whereby a watermark is stored in the state of the software object as it is being run with a particular input sequence.

2. A method as claimed in claim 1 wherein the software object may be a program or piece of program.

3. A method as claimed in any one of claims 1 or 2 wherein the state of the software object may correspond to the current values held in the stack, heap, global variables, registers, program counter and the like.

4. A method as claimed in any preceding claim wherein the watermark is stored in an object's execution state whereby an input sequence I is constructed which, when fed to an application of which the object is a part, will make the object O enter a state which represents the watermark, the representation being validated or checked by examining the stack, heap, global variables, registers, program counter and the like, of the object O.

5. A method as claimed in any one of claims 1 or 2 wherein the watermark is embedded in the execution trace of the object O whereby, as a special input I is fed to O, the address/operator trace is monitored and, based on a property of the trace, a watermark is extracted.

6. A method as claimed in any one of claims 1 to 4 wherein the watermark is embedded in the topology of a dynamically built graph structure.

7. A method as claimed in claim 6 wherein the graph structure (or watermark graph) corresponds to a representation of the data structure of the program and may be viewed as a set of nodes together with a set of vertices.

8.   A method as claimed in any preceding claim further comprising building a recognizer R concurrently with the input I and watermark W.

9.   A method as claimed in claim 8 wherein R is a function adapted to identify and extract the watermark graph from all other dynamic structures on the heap or stack.

10. A method as claimed in either claim 8 or 9 wherein the watermark W incorporates a marker that will allow R to recognize it easily.

11. A method as claimed in any one of claims 8 to 10 wherein R is retained separately from the program and whereby R inspects the state of the program.

12. A method as claimed in any one of claims 8 to 11 wherein R is dynamically linked with the program when it is checked for the existence of a watermark.

13. A method as claimed in any preceding claim wherein the application of which the object forms a part is obfuscated or incorporates tamper-proofing code.

14. A method as claimed in any one of claims 8 to 12 wherein R checks W for a signature property $s(W)$.

15. A method as claimed in claim 14 wherein the signature property $s(W)$ is evaluated by testing for a specific result from a hard computational problem.

16. A method as claimed in either claim 14 or claim 15 including the creation of a number n which may be embedded in the topology of W, whereby the signature property may be evaluated by testing one or more numeric properties of n.

17. A method as claimed in claim 16 wherein the signature property is evaluated by testing whether n is the product of two primes.

00033424

18. A method of verifying the integrity or origin of a program including:
     watermarking the program with a watermark W, wherein the watermark W is stored in the state of a program as the program is being run with a particular input sequence I;
     building a recognizer R concurrently with the input I and watermark W wherein the recognizer is adapted to extract the watermark graph from other dynamically allocated data wherein R is kept separately from the program; wherein R is adapted to check for a number n.

19. A method of verifying the integrity of origin of a program as claimed in claim 18, wherein n is the product of two primes and wherein n is embedded in the topology of W.

20. A method as claimed in either claim 18 or 19 wherein the number n is derived from any combination of numbers depending on the context and application.

21. A method as claimed in any one of claims 18 to 20 wherein the program or code is further adapted to be resistant to tampering, preferably by means of obfuscation or by adding tamper-proofing code.

22. A method as claimed in any one of claims 18 to 21 wherein the recognizer R checks for the effect of the watermarking code on the execution state of the application thereby preserving the ability to recognize the watermark in cases where semantics-preserving transformations have been applied to the application.

23. A method of watermarking software including the steps of:
     embedding a watermark in a static string; and
     applying an obfuscation technique whereby this static string is converted into executable code.

24. A method of watermarking software wherein the watermark W is chosen from a class of graphs G wherein each member of G has one or more properties, such as planarity, said property being capable of being tested by integrity-testing software.

AMENDED SHEET
IPEA/AU

03137157

25. A method of watermarking software as claimed in claim 24 wherein the watermark is rendered tamperproof to certain transformations by subjecting the watermark graph to one or more local transformations.

5

26. A method of watermarking software as claimed in claim 25 wherein each node of the watermark graph is expanded into a cycle.

27. A method of fingerprinting software wherein a plurality of watermarked programs obtained as claimed in any preceding claim are produced.

10

28. A method of fingerprinting software as claimed in claim 27 wherein the watermarked programs each of which has a number n with a common prime factor p.

15

29. A method substantially as herein described with reference to the drawings.

30. Software written to perform the method as claimed in any preceding claim.

20

31. A computer programmed to perform the method as claimed in any one of claims 1 to 27.
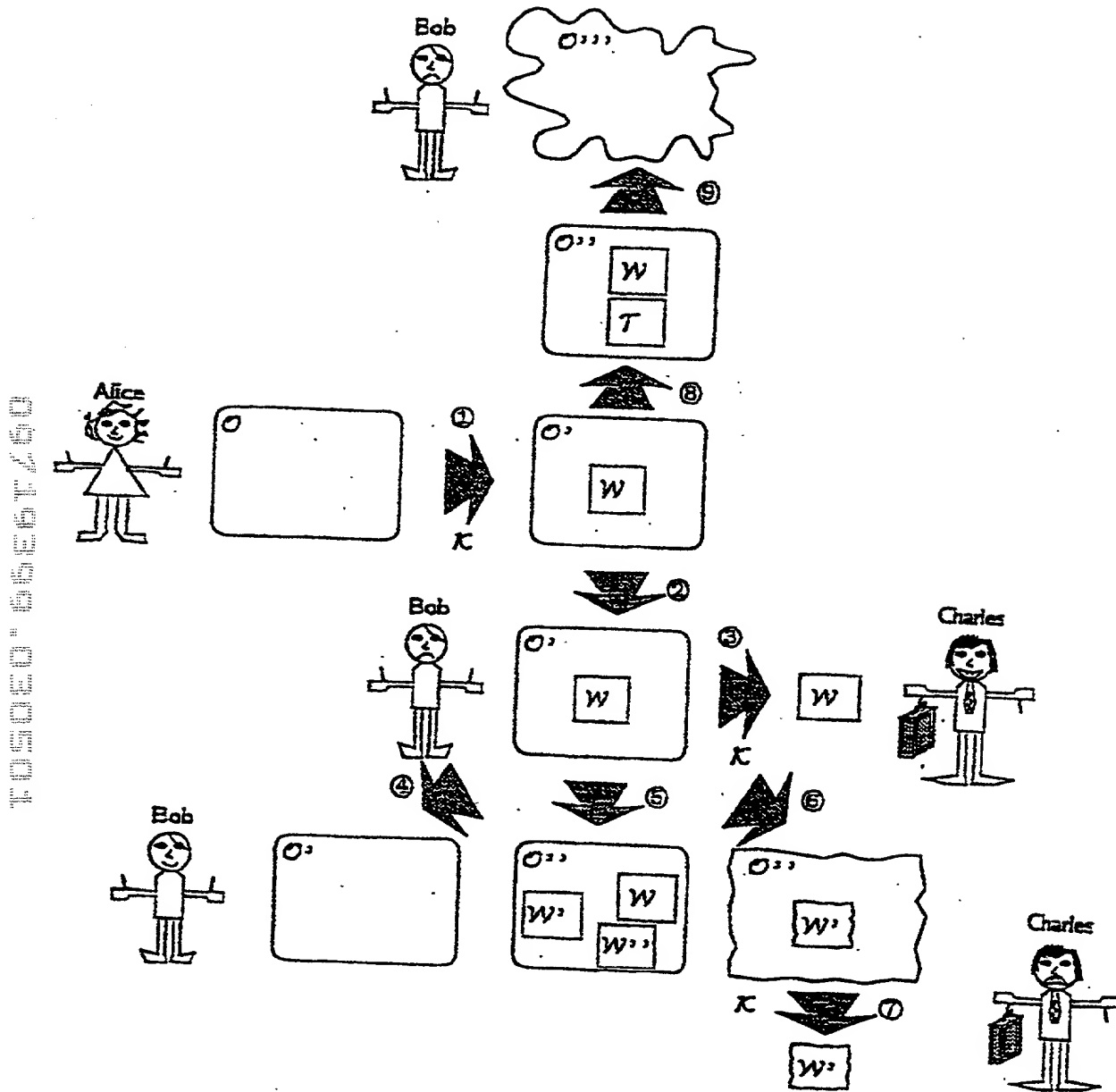
1/11

FIGURE 1

Figure 1: At ① Alice adds a watermark W using key K to her object O to make O'. At ② Bob steals a copy of O'. At ③ Charles extracts the watermark from O' using the key K to show that O' is owned by Alice. At ④ Bob successfully removes W from O. At ⑤ Bob adds new watermarks W' and W'' to make it hard for Charles to prove that W is Alice's original watermark. At ⑥ Bob distorts O' (and W) making it difficult for Charles to detect W. At ⑦ Charles attempts to extract the watermark from the distorted object, and either fails completely or gets a distorted watermark. At ⑧ Alice adds tamperproofing T to O. At ⑨ Bob tries to remove W from O, but, due to the tamperproofing, O will be rendered useless to Bob.
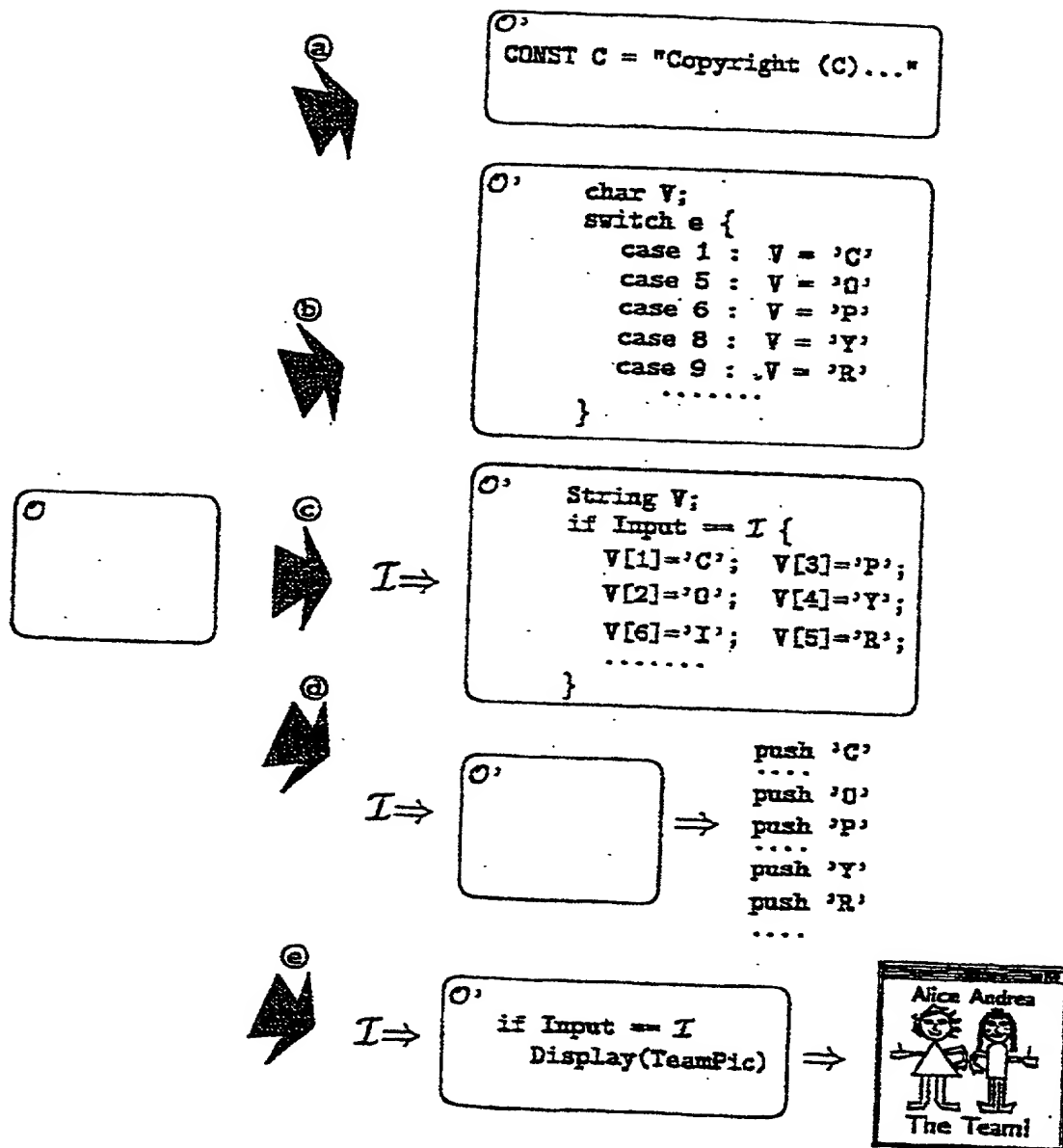
SUBSTITUE SHEET (Rule 26)

```
CONST C = "Copyright (C)..."
```

(a)

```
char V;
switch e {
    case 1 :  V = 'C'
    case 5 :  V = 'O'
    case 6 :  V = 'P'
    case 8 :  V = 'Y'
    case 9 :  V = 'R'
    .......
}
```

(b)

```
String V;
if Input == I {
    V[1]='C';    V[3]='P';
    V[2]='O';    V[4]='Y';
    V[6]='I';    V[5]='R';
    .......
}
```

$\mathcal{I} \Rightarrow$

(c)

(d)

$\mathcal{I} \Rightarrow$

```
push 'C'
....
push 'O'
push 'P'
....
push 'Y'
push 'R'
....
```

$\Rightarrow$

(e)

$\mathcal{I} \Rightarrow$

```
if Input == I
    Display(TeamPic)
```

$\Rightarrow$

Alice Andrea
The Team!

FIGURE 2

Figure 2: In (a) Alice embeds a watermark in the initialized data (string) section of her program. In (b) the watermark is embedded in the text (code) section of the program. In (c) the watermark gets embedded in a global variable V when the program is run with input $\mathcal{I}$. In (d) the watermark is embedded in the execution trace when the program is run with input $\mathcal{I}$. In (e) the watermark is embedded in the unexpected behavior (an "Easter Egg") of the program when it is run with input $\mathcal{I}$.

```
String G (int n) {
    int i=0,k;
    String S;
    while (1) {
        L1:   if (n==1) {S[i++]="A";k=0;goto L6};
        L2:   if (n==2) {S[i++]="B";k=-2;goto L6};
        L3:   if (n==3) {S[i++]="C";goto L9};
        L4:   if (n==4) {S[i++]="X";goto L9};
        L5:   if (n==5) {S[i++]="C";goto L11};
              if (n>12) goto L1;
        L6:   if (k++<=2) {S[i++]="A";goto L6} else goto L8;
        L8:   return S;
        L9:   S[i++]="C"; goto L10;
        L10:  S[i++]="B"; goto L8;
        L11:  S[i++]="C"; goto L12;
        L12:  goto L10;
    }
}
```

FIGURE 3

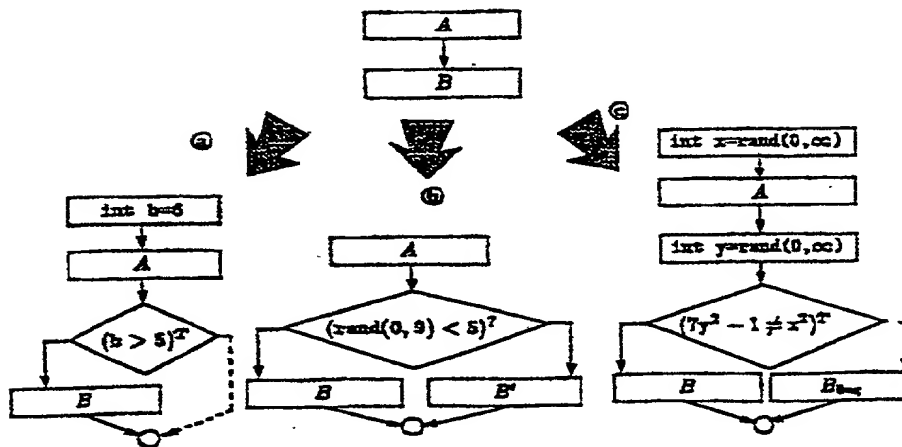Figure 3: A function producing the the strings "AAA", "BAAAA", "XCB", and "CCB".

FIGURE 4

Figure 4: Inserting bogus predicates in a program. In (a) an opaque predicate $b > 5^T$ is inserted. This predicate is always true. In (b) an opaque predicate $rand(0,9) < 5^T$ is inserted. This predicate is sometimes true (in which case B is executed), and sometimes false (in which case an obfuscated version of B is executed). In (c) an opaque true predicate is inserted. This predicate appears to sometimes execute an obfuscated buggy version of B, but, in fact, never does.

SUBSTITUE SHEET (Rule 26)

$g(V)$		$f(p,q)$	
p	q	V	$2p+q$
0	0	False	0
0	1	True	1
1	0	True	2
1	1	False	3

	AND[A,B]	0	1	2	3
	0	3	0	0	0
B	1	3	1	2	3
	2	0	2	1	3
	3	3	0	0	3

(with column header **A** spanning 0 1 2 3)

```
(1) bool A,B,C;              (1') short a1,a2,b1,b2,c1,c2;
(2) B = False;               (2') b1=0; b2=0;
(3) C = False;          T    (3') c1=1; c2=1;
(4) C = A & B;         =>    (4') x=AND[2*a1+a2,2*b1+b2]; c1=x/2; c2=x%2;
(5) C = A & B;               (5') c1=(a1 ^ a2) & (b1 ^ b2); c2=0;
(6) if (A) ...;              (6') x=2*a1+a2; if ((x==1) || (x==2)) ...;
(7) if (B) ...;              (7') if (b1 ^ b2) ...;
```

FIGURE 5

Figure 5: Variable splitting example. We show one possible choice of representation for split boolean variables. The table indicates that boolean variable V has been split into two short integer variables p and q. If $p = q = 0$ or $p = q = 1$ then V is False, otherwise, V is True. Given this new representation, we devise substitutions for the built-in boolean operations. In the example, we provide a run-time lookup table for each operator. Given two boolean variables $V_1 = [p,q]$ and $V_2 = [r,s]$, $\ulcorner V_1 \& V_2 \urcorner$ is computed as $\ulcorner \text{AND}[2p+q, 2r+s] \urcorner$.

$$Z(X+r,Y) = 2^{32} \cdot Y + (r+X) = Z(X,Y) + r$$
$$Z(X,Y+r) = 2^{32} \cdot (Y+r) + X = Z(X,Y) + r \cdot 2^{32}$$
$$Z(X \cdot r,Y) = 2^{32} \cdot Y + X \cdot r = Z(X,Y) + (r-1) \cdot X$$
$$Z(X,Y \cdot r) = 2^{32} \cdot Y \cdot r + X = Z(X,Y) + (r-1) \cdot 2^{32} \cdot Y$$

```
(1) int X=45;                (1') long Z=167759086119551045;
    int Y=95;
(2) X += 5;             T    (2') Z += 5;
(3) Y += 11;           =>    (3') Z += 47244640256;
(4) X *= c;                  (4') Z += (c-1)*(Z & 4294967295);
(5) Y *= d;                  (5') Z += (d-1)*(Z & 18446744069414584320);
```

FIGURE 6

Figure 6: Merging two 32-bit variables X and Y into one 64-bit variable Z. Y occupies the top 32 bits of Z, X the bottom 32 bits. If the actual range of either X or Y can be deduced from the program, less intuitive merges could be used. First we give rules for addition and multiplication with X and Y, then show some simple examples.

5/11

```
int Sum(int A[]) {
  int i, sum=0;
  int n=A.length;
  for (i=0;i<n;i++)
    sum += A[i];
  return sum;
}
```

$\xrightarrow{\;T\;}$

```
int Sum(int A[]) {
  int sum=0, i=0, pc=0;
  int s[]=new int[5], sp=-1;
  loop: while (true)
    switch("fcgabced".charAt(pc)) {
      case 'a':  sum += s[sp--]; pc++; break;
      case 'b':  i++; pc++; break;
      case 'c':  s[++sp] = i; pc++; break;
      case 'd':  if (s[sp--] > s[sp--]) pc -= 6;
                 else break loop; break;
      case 'e':  s[++sp] = A.length; pc++; break;
      case 'f':  pc += 5; break;
      case 'g':  s[sp] = A[s[sp]]; pc++; break;
    }
  return sum;
}
```

FIGURE 7

Figure 7: The Java method Sum on the left is obfuscated by translating it into the bytecode "fcgabced". This code is then executed by a stack-based interpreter specialized to handle this particular virtual machine code. This technique is similar to Proebsting's superoperators [20].
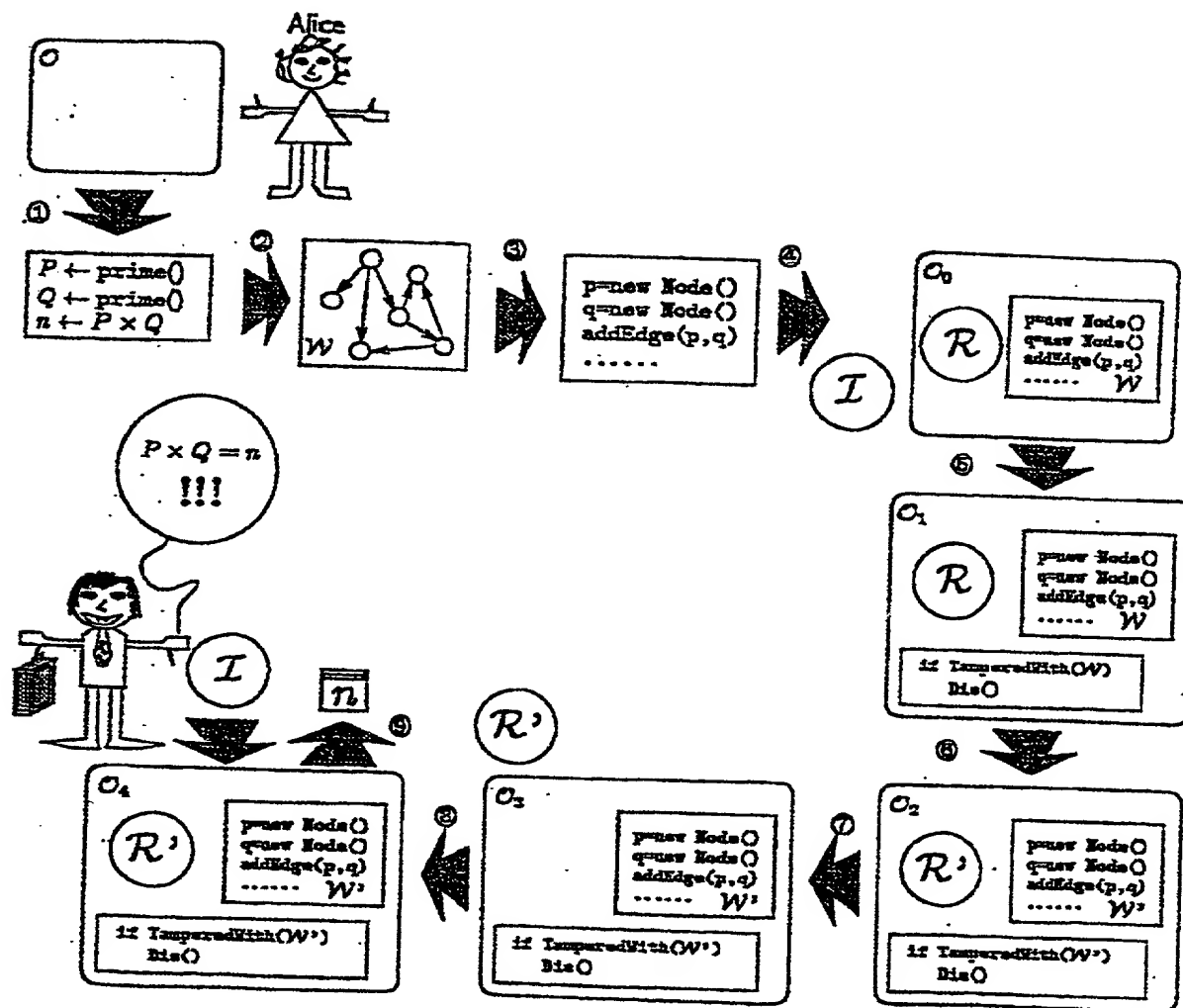
FIGURE 8

Figure 8: At ① Alice selects two large primes P and Q, and computes their product n. At ② she embeds n in the topology of a graph. This graph is her watermark $\mathcal{W}$. At ③ $\mathcal{W}$ is converted to a program which builds the graph. At ④ the program is embedded into the original program $\mathcal{O}$, such that when $\mathcal{O}_0$ is run with $\mathcal{I}$ as input, $\mathcal{W}$ is built. Also, a recognizer program $\mathcal{R}$ is constructed, which is able to identify $\mathcal{W}$ on the heap, and extract n from it. At ⑤ tamperproofing is added, to prevent the graph from being obfuscated to such an extent that $\mathcal{R}$ cannot identify it. At ⑥ the application (including the watermark, tamperproofing code, and recognizer) is obfuscated. At ⑦ the recognizer is removed from the application. $\mathcal{O}_3$ is the version of Alice's program that is distributed. At ⑧ Charles links in the recognizer program $\mathcal{R}$ with $\mathcal{O}_3$. At ⑨ the application is run with $\mathcal{I}$ as input, and the recognizer $\mathcal{R}$ produces n. Since Charles is the only one who can factor n, he can prove the legal origin of Alice's program.

SUBSTITUE SHEET (Rule 26)

$$3 \cdot 6^4 \quad + \quad 2 \cdot 6^3 \quad + \quad 3 \cdot 6^2 \quad + \quad 4 \cdot 6^1 \quad + \quad 1 \cdot 6^0 = 4453 = 61 * 73$$
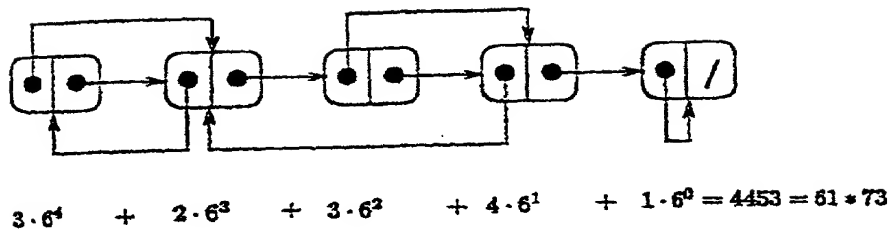
FIGURE 9

Figure 9: Embedding a watermark into a graph structure. The structure is essentially a linked list. The rightmost pointer of each node is the next field, while the second field encodes a digit. In this example, 0=null (/), 1=a self-pointer, 2=a one-step back pointer, 3=a one step forward pointer, 4=a two step back pointer, and 5=a 2 step forward pointer. This allows us to encode a value $61 * 73 = 4453_{10}$ as the base-6 value 32341_6.
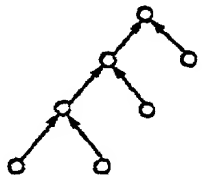

FIGURE 10

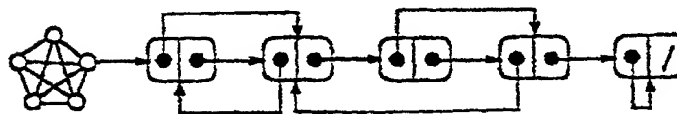Figure 10: The twenty-second tree in an enumeration of the oriented trees with seven vertices.

FIGURE 11

Figure 11: A 5-clique is used to mark the beginning of an encoded value.
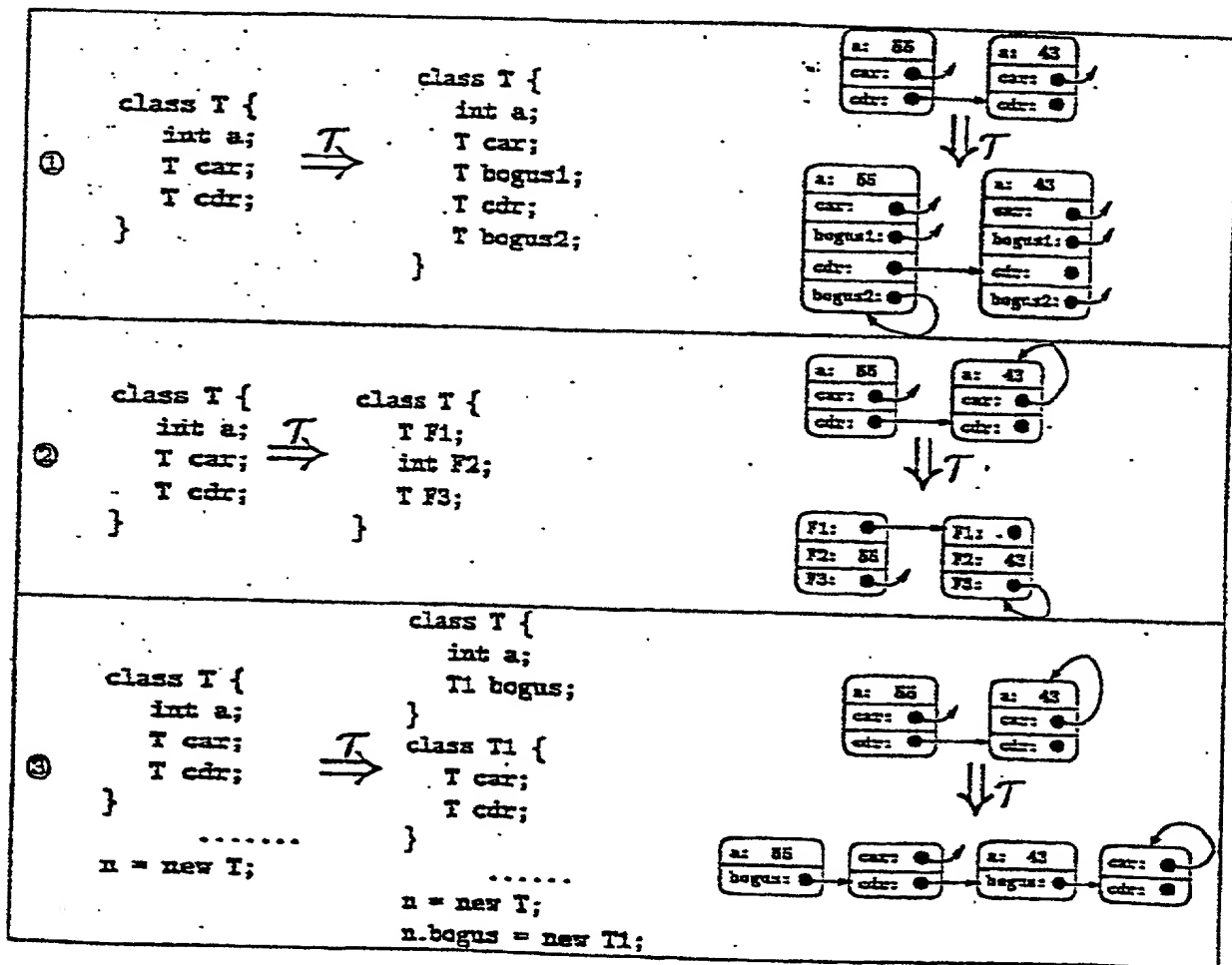
SUBSTITUE SHEET (Rule 26)

FIGURE 12

Figure 12: Obfuscation of dynamic structures. In ① we add bogus pointer fields to all nodes of type T. In ② we rename and reorder fields. In ③ we add a level of indirection by splitting all nodes in two.

```
class C {public int a; public C car, cdr;}

public static void main(String[] args) {
    Field[] F = C.class.getFields();
    if (F.length != 3)
        die();
    if (F[0].getType() !=
        java.lang.Integer.TYPE)
        die();
    if (F[1].getType() != C.class)
        die();
    if (F[2].getType() != C.class)
        die();
}
```

(a)

```
class C {public int a; public C car, cdr;}

public static void main(String[] args)
        throws NoSuchFieldException,
        IllegalAccessException {
    Field f;
    String V;
    C n = new C();
    Class c = n.getClass();
    if (P^F) {
        f = c.getField(V="car");
        ① f.set(n, null);
    }

    Field F = c.getFields();
    int R;
    ② F[R="1"].set(n, n.car);
}
```

(b)

FIGURE 13

Figure 13: Examples of tamperproofing Java code using the reflection interface.
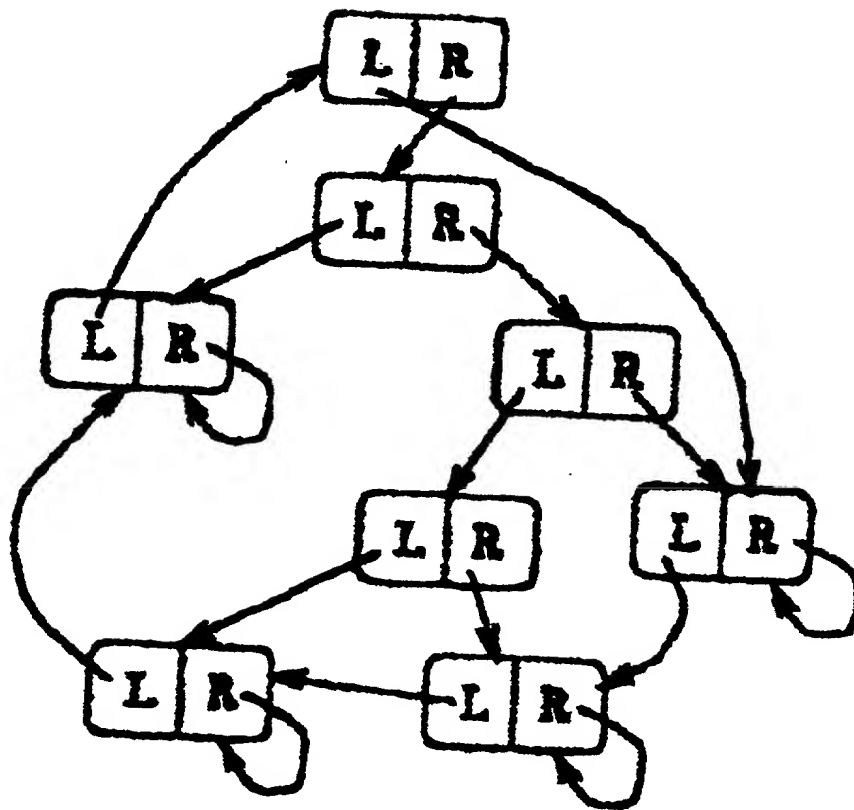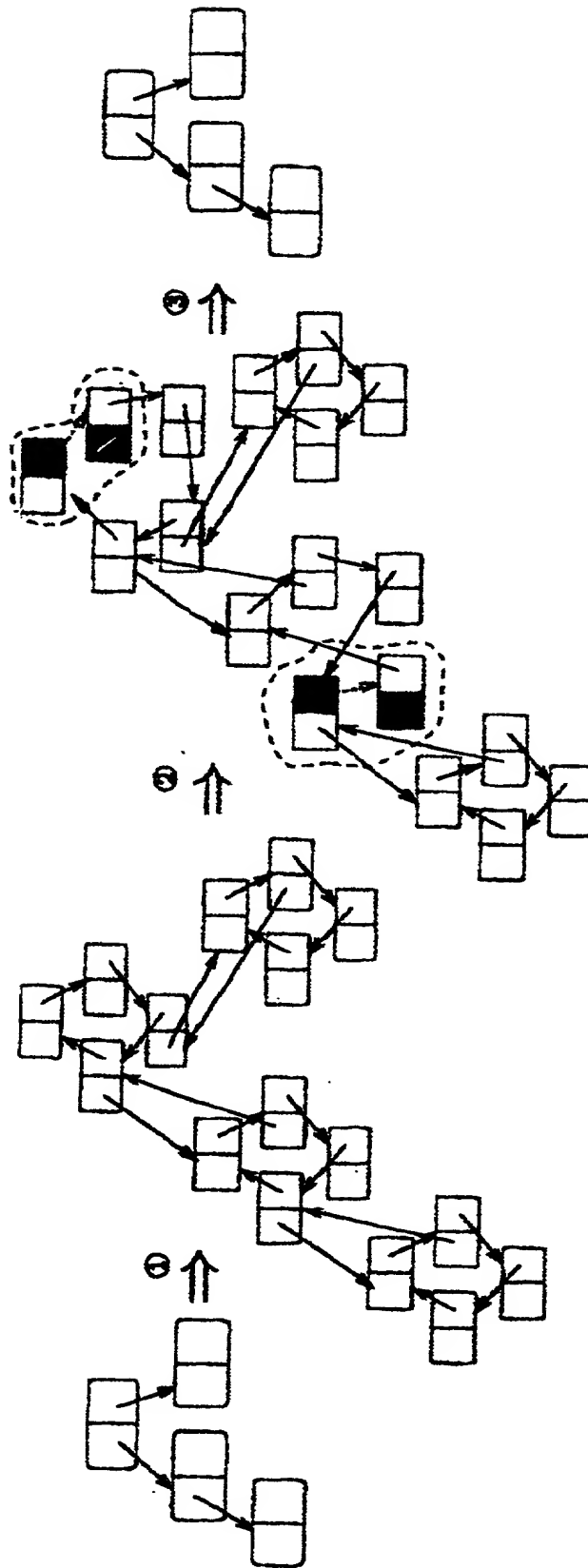
FIG 14

Figure 15 Tamperproofing against node-splitting. At ① we expand each node of our original watermark tree into a 4-cycle. At ② an adversary splits two nodes. The structure of the graph ensures that these nodes will fall on a cycle. At ③ the recognizer shrinks the biconnected components of the underlying (undirected) graph. The result is a graph isomorphic to our original watermark.

FIG 15

DECLARATION AND POWER OF ATTORNEY FOR **UTILITY** PATENT APPLICATION

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below, next to my name,

I believe that I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled

SOFTWARE WATERMARKING TECHNIQUES

the specification of which

　　　___ is attached hereto.
　　　_X_ was assigned Ser. No. 09/719,399 and International Filing date June 10, 1999.

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above. I do not know and do not believe that the same was ever known or used in the United States of America before my invention thereof, or patented or described in any printed publication in any country before my invention thereof or more than one year prior to this application, that the same was not in public use or on sale in the United States of America more than one year prior to this application, and said invention has not been patented or made the subject of an inventor's certificate issued before the date of this application in any country foreign to the United States of America on an application filed by me or my legal representatives or assigns more than twelve months prior to this application.

I acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, Code of Federal Regulations, Section 1.56 (a).

I hereby claim foreign priority benefits under Title 35, United States Code, Section 119, of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s)　　　　　　　　　　　　　　　　　　　　　Priority Claimed

PCT/NZ99/00081	PCT	10 June 1999 (10.06.99)	X	
(Number)	(Country)	(Day/Month/Year Filed)	Yes	No

330675	NZ	10 June 1998 (10.06.98)	X	
(Number)	(Country)	(Day/Month/Year Filed)	Yes	No

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

(Application Serial No.)	(Filing Date)	(Status -- patented, pending, abandoned)

(Application Serial No.)	(Filing Date)	(Status -- patented, pending, abandoned)

(4)

I hereby appoint Joseph A. Sawyer, Jr., Reg. No. 30,801; Stephen G. Sullivan, Reg. No. 38,329; Janyce R. Mitchell, Reg. No. 40,095; and Michele Liu, Reg. No.: 44,875; of SAWYER LAW GROUP LLP, located at 2465 E. Bayshore Rd., Suite 406, Palo Alto, California 94303, telephone (650) 493-4540, as my attorneys with full power of substitution and revocation, to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith.

Address all telephone calls to Mr. Joseph A. Sawyer, Jr. at telephone number (650) 493-4540 and all correspondence to:

Joseph A. Sawyer, Jr.
SAWYER & ASSOCIATES
P.O. Box 51418
Palo Alto, California 94303

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of first/joint Inventor: **Christian Sven Collberg**

Residence Address: **7818 E. Pristine Pl,**
                                       Tucson, Arizona 85750 *AZ*

Post Office Address: SAME

Country of Citizenship: **Sweden**

Jan 24, 2001   _(signature)_

Date   Signature

Full Name of second/joint Inventor: **Clark David Thomborson**

Residence Address: **3/61 Fancourt Street**
                                       Meadowbanks
                                       Auckland
                                       New Zealand *NZL*

Post Office Address: SAME

Country of Citizenship: **United States of America**

12 FEBRUARY 2001   _(signature)_

Date   Signature